So Wrong

Absurd ways of doing perfectly normal things

Why?

- We already know how to do these things
- By doing them in strange ways, maybe we can learn something new?

Tell someone they owe you ₹ 819

Tell someone they owe you ₹ 819

- But you have to count on your hands

- No pointing to numbers
- No writing numbers on your palm

1 1 0 0 1 1 0 0 1 1

1100110011 = 819

Let’s do the same thing in Python

Web Frameworks

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Web Frameworks

```
from flask.views import View

class ShowUsers(View):

    def dispatch_request(self):
        users = User.query.all()
        return render_template('users.html', objects=users)

app.add_url_rule('/users/', view_func=ShowUsers.as_view('show_users'))
```

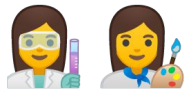
Web Frameworks

```
from flask.views import View

class ShowUsers(View):

    def dispatch_request(self):
        users = User.query.all()
        return render_template('users.html', objects=users)

app.add_url_rule('/users/', view_func=ShowUsers.as_view('show_users'))
```

- What if you cannot use CLASSES and FUNCTIONS

How about this instead?

```
from sowrong import GET, run

async for request, response in GET("/"):
    response.body = "hi"
```

PPTs are fine. Show me the code 

Everything must eventually become SQL

```
>>> all_entries = Entry.objects.all()
```

```
# Example of iterating over the results of a query using the cursor.
cursor = db.execute_sql('SELECT * FROM users WHERE status = ?', (ACTIVE,))
```

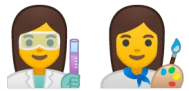
Everything must eventually become SQL

```
>>> all_entries = Entry.objects.all()
```

```
# Example of iterating over the results of a query using the cursor.
cursor = db.execute_sql('SELECT * FROM users WHERE status = ?', (ACTIVE,))
```

- Can we do it without
  - Custom classes for each table/keyword etc?
  - Strings

```
def create_user_table():
    sql: create . table . user ( name . text, age . real)

def insert_values():
    sql: insert . into . user . values ('arjoonn', 600)

def show_table():
    sql: select * frm . user
```

Code 

Stalling APIs are bad. Offload to another process

```
from celery import Celery

app = Celery('tasks', broker='pyamqp://guest@localhost//')

@app.task
def add(x, y):
    return x + y
```

Stalling APIs are bad. Offload to another process

```
from celery import Celery

app = Celery('tasks', broker='pyamqp://guest@localhost//')

@app.task
def add(x, y):
    return x + y
```

- But what if you HAD to stay in the same process?
- Cannot change to async server
- Cannot change the person who is calling your API

```
@app.get("/do/<x>")
def do(x):
    big_task(x)
```

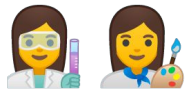
```
@app.get("/do/<x>")
def do(x):
    big_task(x)
```

```
@batch(app.get("/do/<x>"))
def do(x):
    yield from big_task(x)
```

```python
@app.get("/do/<x>")
def do(x):
    big_task(x)
```

```python
@batch(app.get("/do/<x>"))
def do(x):
    yield from big_task(x)
```

```
curl -L --max-redirs -1 http://localhost:8080/do/100000
```

Code 

So what do we learn?

Clear thought leads to exposed faults

```
async for request, response in GET("/"):
    response.body = "hi"
```

Clear thought leads to exposed faults

```
async for request, response in GET("/"):
    response.body = "hi"
```

- You cannot do an early exit
- You cannot decorate a loop

Things that are forced, break down faster

```
def create_user_table():
    sql: create . table . user ( name . text, age . real)

def insert_values():
    sql: insert . into . user . values ('arjoonn', 600)

def show_table():
    sql: select * frm . user
```

Things that are forced, break down faster

```
def create_user_table():
    sql: create . table . user ( name . text, age . real)

def insert_values():
    sql: insert . into . user . values ('arjoonn', 600)

def show_table():
    sql: select * frm . user
```

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

A good solution solves all cases. Not special ones

```
@batch(app.get("/do/<x>"))
def do(x):
    yield from big_task(x)
```

A good solution solves all cases. Not special ones

```
@batch(app.get("/do/<x>"))
def do(x):
    yield from big_task(x)
```

- Not all long running tasks can be broken into batches
- You truly CANNOT serve another request while a batch item is running

Questions?

- Other things to check out
- https://github.com/satwikkansal/wtfpython
- http://entrian.com/goto/
- https://github.com/thesage21/SoWrong

- Arjoonn.com
- github.com/thesage21